Exercice 1. *Hasta la vistar*

On considère un groupe de n personnes. Chaque personne $i \in [\![0, n-1]\!]$ peut connaître ou non chaque autre personne $j \in [\![0, n-1]\!]$ (avec $i \neq j$).

On appelle $\star$***Star***$\star$ une personne qui ne connaît personne, mais que tout le monde connaît.

On s'intéresse au problème suivant :

Étant donné un groupe de n personnes, contient-il une $\star$***Star***$\star$?

Pour résoudre ce problème, on considère des algorithmes qui peuvent poser des questions du type "***Est ce que*** i ***connaît*** j ***?***".

On note $i \to j$ une telle question, et on compte la complexité d'un algorithme en fonction du nombre de questions qu'il pose.

Q1. Combien peut-il y avoir de stars dans un groupe ?

Q2. Comment modéliser cette situation ? Autrement dit, quelle(s) structure(s) simple(s) peut-on utiliser pour représenter les données du problème ? Avec cette modélisation, à quoi correspond une question ?

Q3. Donner un algorithme en $\mathcal{O}(n^2)$ permettant de trouver une star dans un groupe.

Q4. On considère une question $i \to j$ avec $i, j \in [\![0, n-1]\!]$. Montrer que quelle que soit la réponse, on peut éliminer i ou j comme potentielles $\star$***Stars***$\star$.

Q5. Donner un algorithme en $\mathcal{O}(n)$ permettant de trouver une star dans un groupe.

Q6. Dans le pire cas, quel est le nombre **exact** de questions posées par votre algorithme, en fonction de n ?

Q7. Trouver un algorithme permettant de trouver une star en au plus $3n - \lfloor \log(n) \rfloor - 3$ questions.

On peut montrer qu'il n'existe pas d'algorithme posant strictement moins de questions !

Exercice 2. *Tri à bulle*

Étudier la correction et la complexité de l'algorithme suivant :

Algorithme 1 : Tri à bulle

Entrée(s) : T tableau de taille n
Sortie(s) : Rien, T est trié dans l'ordre croissant

```
1 pour i = 0 à n − 1 faire
2     pour j = 0 à n − 2 faire
3         si T[j] > T[j + 1] alors
4             Échanger T[j] et T[j + 1];
```

Exercice 3. *Occurrences*

Q1. Proposer un algorithme quadratique permettant de déterminer l'élément ayant le plus d'occurrences dans un tableau T donné.

Q2. Proposer une amélioration simple de l'algorithme précédent réduisant la complexité à $\mathcal{O}(n \log n)$.

Soit $K \in \mathbb{N}$. On dit qu'un tableau T est **borné par** K si ses valeurs sont comprises entre 0 et $K-1$. Étant donné un tableau T borné par K de taille n , On appelle **tableau des occurrences** le tableau U de taille K tel que :

$$\forall k \in [\![0, K-1]\!], U[k] = \textbf{card}(\{i \in [\![0, n-1]\!] \mid T[i] = k\}$$

i.e. dont la case k contient le nombre d'occurrences de k dans U.

Q3. Pour $K = 7$, donner le tableau des occurrences de $[5, 0, 2, 2, 1, 3, 5, 1]$.

Q4. Proposer un algorithme en $\mathcal{O}(N)$ permettant de construire le tableau des occurrences d'un tableau T borné par K de taille N. Ce tableau prendra en entrée T ainsi que l'entier K, et devra vérifier que T est bien borné par K.

Q5. En utilisant le tableau des occurrences, proposer un algorithme en $\mathcal{O}(N + K)$ permettant de déterminer l'élément ayant le plus d'occurrences dans un tableau T borné par K de taille N.

Q6. Proposer un algorithme permettant de trier un tableau T borné par K de taille N en temps $\mathcal{O}(N + K)$.